# **JAVASCRIPT**

**MỤC LỤC**

[**JS CORE** 3](#_Toc66523205)

[**1. Built-in function** 4](#_Toc66523206)

[**1.1. Alert** 4](#_Toc66523207)

[**1.2. Console** 4](#_Toc66523208)

[**1.3. Confirm** 4](#_Toc66523209)

[**1.4. Prompt** 5](#_Toc66523210)

[**1.5. Set timeout** 5](#_Toc66523211)

[**1.6. Set Interval** 5](#_Toc66523212)

[**2. Operator** 6](#_Toc66523213)

[**2.1. Toán tử số học** 6](#_Toc66523214)

[**2.2. Toán tử gán** 8](#_Toc66523215)

[**2.3. Toán tử chuỗi** 8](#_Toc66523216)

[**2.4. Toán tử so sánh** 9](#_Toc66523217)

[**2.5. Toán tử logic** 10](#_Toc66523218)

[**3. Boolean & If-else** 12](#_Toc66523219)

[**3.1. Boolean** 12](#_Toc66523220)

[**3.2. If-else** 12](#_Toc66523221)

[**4. Datatype** 14](#_Toc66523222)

[**4.1. Dữ liệu nguyên thuỷ** 14](#_Toc66523223)

[**4.2. Dữ liệu phức tạp** 15](#_Toc66523224)

[**5. String** 17](#_Toc66523225)

[**5.1. Khái niệm về chuỗi** 17](#_Toc66523226)

[**5.2. Làm việc với chuỗi** 18](#_Toc66523227)

[**6. Number** 22](#_Toc66523228)

[**6.1. Khái niệm về số** 22](#_Toc66523229)

[**6.2. Làm việc với số** 22](#_Toc66523230)

[**7. Array** 24](#_Toc66523231)

[**7.1. Khái niệm về mảng** 24](#_Toc66523232)

[**7.2. Làm việc với mảng** 25](#_Toc66523233)

[**7.3. Array method** 28](#_Toc66523234)

[**8. Function** 34](#_Toc66523235)

[**8.1. Tính chất về hàm** 34](#_Toc66523236)

[**8.2. Tham số và đối số hàm** 34](#_Toc66523237)

[**8.3. Arguments** 34](#_Toc66523238)

[**8.5. Các loại hàm** 35](#_Toc66523239)

[**9. Object** 37](#_Toc66523240)

[**9.1. Khái niệm về Object** 37](#_Toc66523241)

[**9.2. Object Constructor** 38](#_Toc66523242)

[**9.3. Object Prototype** 38](#_Toc66523243)

[**10. Date** 39](#_Toc66523244)

[**10.1. Khái niệm về Date** 39](#_Toc66523245)

[**10.2. Sử dụng Date object** 39](#_Toc66523246)

[**11. Branch Command** 40](#_Toc66523247)

[**11.1. If-else** 40](#_Toc66523248)

[**11.2. Switch-case** 40](#_Toc66523249)

[**12. Loop** 42](#_Toc66523250)

[**12.1. Vòng lặp for** 42](#_Toc66523251)

[**12.2. Vòng lặp while** 42](#_Toc66523252)

[**12.3. Vòng lặp do-while** 43](#_Toc66523253)

[**12.4. Vòng lặp for-in** 44](#_Toc66523254)

[**12.5. Vòng lặp for-of** 45](#_Toc66523255)

[**12.6. Vòng lặp lồng vòng lặp (nested loop)** 46](#_Toc66523256)

[**13. Callback** 47](#_Toc66523257)

[**13.1. Khái niệm callback** 47](#_Toc66523258)

[**13.2. Hiểu thêm cách hoạt động của callback** 47](#_Toc66523259)

[**14. JSON** 48](#_Toc66523260)

[**14.1. Khái niệm về JSON** 48](#_Toc66523261)

[**14.2. Sử dụng đối tượng JSON** 48](#_Toc66523262)

[**15. Promise** 50](#_Toc66523263)

[**15.1. Khái niệm về Promise** 50](#_Toc66523264)

[**15.2. Pain** 51](#_Toc66523265)

[**15.3. Concept** 52](#_Toc66523266)

[**15.4. Chain** 53](#_Toc66523267)

[**16. Fetch** 54](#_Toc66523268)

[**16.1. Khái niệm về fetch API** 54](#_Toc66523269)

[**16.2. Sử dụng fetch** 54](#_Toc66523270)

[**ES6** 55](#_Toc66523271)

[**1. Let & Const** 56](#_Toc66523272)

[**1.1 Tìm hiểu về khai báo biến** 56](#_Toc66523273)

[**1.2. Sự khác nhau** 56](#_Toc66523274)

[**2. Arrow Function** 58](#_Toc66523275)

[**3. Classes** 59](#_Toc66523276)

[**4. Enhanced Object literals** 60](#_Toc66523277)

[**5. Destructuring** 61](#_Toc66523278)

[**5.1. Khái niệm về destructuring** 61](#_Toc66523279)

[**5.2. Sử dụng destructuring** 61](#_Toc66523280)

[**6. Spread** 63](#_Toc66523281)

[**6.1. Khái niệm về spread** 63](#_Toc66523282)

[**6.2. Sử dụng spread** 63](#_Toc66523283)

[**DOM HTML** 64](#_Toc66523284)

[**1. DOM Object** 65](#_Toc66523285)

[**1.1. Element** 65](#_Toc66523286)

[**1.2. Attribute** 67](#_Toc66523287)

[**1.3. Text** 68](#_Toc66523288)

# **JS CORE**

## **1. Built-in function**

Giới thiệu 1 số hàm built-in:

1. Alert
2. Console
3. Confirm
4. Prompt
5. Set timeout
6. Set Interval

Cụ thể như sau:

### **1.1. Alert**



Trình duyệt sẽ tạo 1 message có nội dung “test xem cái alert như nào” và 1 button “OK”

### **1.2. Console**



Đây là hàm built-in mà 1 dev sẽ sử dụng xuyên suốt quá trình code JS. Kiểm tra bằng cách vào browser nhấn F12 -> tab console.

### **1.3. Confirm**



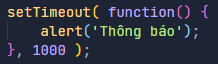
Tương tự như alert, trình duyệt sẽ tạo 1 message có nội dung “test” nhưng ở đây sẽ có 2 button là “OK” và “Cancel” và chức năng của 2 button này sẽ khác nhau thông qua cách định nghĩa của dev.

### **1.4. Prompt**



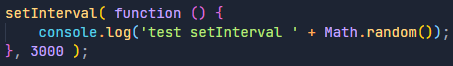
Tương tự như confirm, trình duyệt sẽ tạo 1 message có nội dung “Xác nhận tên của bạn” có thêm 2 button “OK” và “Cancel” tuy nhiên nó sẽ tạo thêm cho chúng ta 1 ô input để nhập vào, và tuỳ theo cách dev định nghĩa.

### **1.5. Set timeout**



hàm setTimeout sẽ thực hiện 1 đoạn code trong declaration function sau 1 khoảng thời gian ấn định, lưu ý chỉ chạy 1 lần.

### **1.6. Set Interval**



Hàm setInterval sẽ thực hiện 1 đoạn code trong declaration function sau 1 khoảng thời gian ấn định, nó sẽ chạy n lần sau 1 khoảng thời gian chúng ta set.

## **2. Operator**

Toán tử trong JS gồm:

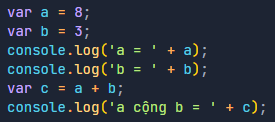
1. Số học
2. Gán
3. Chuỗi
4. So sánh
5. Logic

Phân tích sâu vào các toán tử của JS

### **2.1. Toán tử số học**

|  |  |
| --- | --- |
| **Phép toán** | **Tên toán tử** |
| + | Cộng |
| - | Trừ |
| \* | Nhân |
| \*\* | Luỹ thừa |
| / | Chia |
| % | Chia lấy dư |
| ++ | Tăng 1 giá trị số |
| -- | Giảm 1 giá trị số |

4 toán tử cơ bản ( + - \* / )



Kết quả trả lại sẽ là: 

Luỹ thừa ( \*\* )



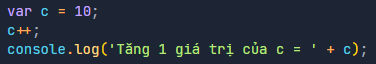
Kết quả trả lại sẽ là: 

Chia lấy dư ( % )



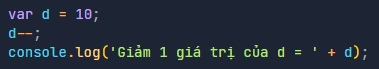
Kết quả trả lại sẽ là: 

Tăng 1 giá trị số ( ++ )



Kết quả trả lại sẽ là: 

Giảm 1 giá trị số ( -- )

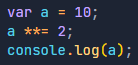


Kết quả trả lại sẽ là: 

### **2.2. Toán tử gán**

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Ví dụ** | **Tương đương** |
| **=** | x = y | x = y |
| **+=** | x += y | x = x + y |
| **-=** | x -= y | x = x – y |
| **\*=** | x \*= y | x = x \* y |
| **/=** | x /= y | x = x / y |
| **\*\*=** | x \*\*= y | x = x \*\* y |

Gán số



Kết quả trả lại sẽ là: 

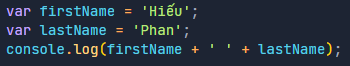
### **2.3. Toán tử chuỗi**



Kết quả trả lại sẽ là: 

**###** Vậy chúng ta sẽ nối chuỗi như thế nào ???

Cách 1:



Kết quả trả lại sẽ là: 

Cách 2:

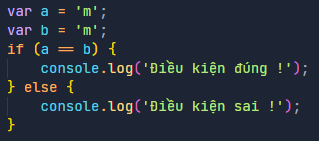
Chúng ta sẽ áp dụng kiến thức cũ bằng cách dùng toán tử gán như sau:



Kết quả trả lại sẽ vẫn nỗi chuỗi như cách 1: 

### **2.4. Toán tử so sánh**

|  |  |
| --- | --- |
| **Toán tử** | **Tên toán tử** |
| **==** | Bằng |
| **!=** | Không bằng |
| **>** | Lớn hơn |
| **<** | Nhỏ hơn |
| **>=** | Lớn hơn hoặc bằng |
| **<=** | Bé hơn hoặc bằng |



Kết quả trả lại sẽ là: 

Tuy nhiên toán tử so sánh vẫn còn 2 phép toán đòi hỏi sự tuyệt đối đến datatype.

|  |  |
| --- | --- |
| **Toán tử** | **Tên toán tử** |
| **===** | Bằng |
| **!==** | Không bằng |

2 toán tử bằng và không bằng trên có lẻ giống nhau. Cho nên chúng ta sẽ có 1 ví dụ để chứng minh sự khác nhau giữa chúng.



*Ví dụ hình ảnh*

Đối với toán tử ( == )

**

Kết quả trả lại sẽ là: 

Đối với toán tử ( === )

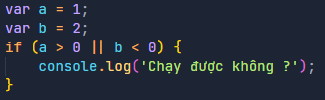
**

Kết quả trả lại sẽ là: 

Từ 2 kết quả trên cho thấy toán tử ( === ) sẽ cho so sánh cả datatype. Vì thế kết quả trả lại sẽ là 

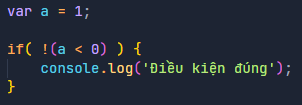
### **2.5. Toán tử logic**

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Tên toán tử** | **Ý nghĩa** |
| && | AND | Tất cả các điều kiện phải đúng |
| || | OR | Chỉ cần 1 điều kiện đúng |
| ! | NOT | Phủ định |

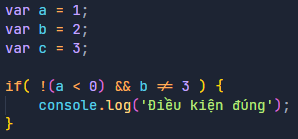


Kết quả trả lại sẽ là: 

Vì là toán tử || nên chỉ cần 1 trong 2 điều kiện đúng thì lệnh sẽ thực hiện.



Kết quả trả lại sẽ là: 



Kết quả trả lại sẽ là: 

Vì là toán tử && nên tất cả điều kiện phải đúng thì lệnh mới thực hiện.

## **3. Boolean & If-else**

### **3.1. Boolean**

Boolean trong JS mô tả 2 giá trị là: true và false

**

Kết quả trả lại sẽ là: 

### Hiểu rõ hơn về Boolean ( falsy – truthy )

Trong JS có 6 giá trị sau là falsy:

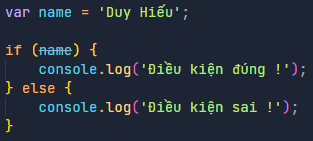
* 0
* false
* ‘’
* undefined
* NaN
* null

và khi convert sang boolean thì nó sẽ có giá trị là false.

Còn lại tất cả giá trị khác đều là truthy. Kể cả những giá trị đặc biệt sau:

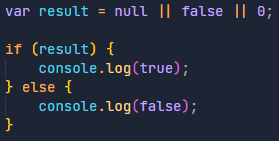
* ‘0’
* ‘false’
* []
* {}
* function() {}

### **3.2. If-else**

**

Kết quả trả lại sẽ là: 

Ví dụ rõ hơn về boolean + if-else



Kết quả trả lại sẽ là: 

Biến result trong lúc này sẽ nhận giá trị là 0 thông quan toán tử || . Mà 0 lúc này sẽ mang giá trị là falsy cho nên result = false. Qua đó kết quả trả lại sẽ là 

## **4. Datatype**

Kiểu dữ liệu trong JS:

1. Dữ liệu nguyên thuỷ
   * Number
   * String
   * Boolean
   * Undefined
   * Symbol
2. Dữ liệu phức tạp
   * Function
   * Object
   * Array

Tìm hiểu cụ thể về những kiểu dữ liệu trên

### **4.1. Dữ liệu nguyên thuỷ**

#### **4.1.1. Number**



Kết quả là: 

#### **4.1.2. String**



Kết quả là: 

#### **4.1.3. Boolean**



Kết quả là: 

#### **4.1.4. Undefined**



Kết quả là: 

#### **4.1.5. Null**



Kết quả là: 

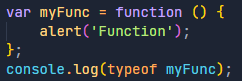
#### **4.1.6. Symbol**



Kết quả là: 

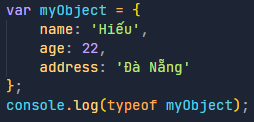
### **4.2. Dữ liệu phức tạp**

#### **4.2.1. Function**



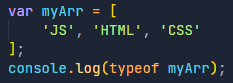
Kết quả là: 

#### **4.2.2. Object**



Kết quả là: 

#### **4.2.3. Array**



Kết quả là: 

Chúng ta sử dụng typeof + tên biến để kiểm tra kiểu dữ liệu

### Object và array đều là kiểu dữ liệu Object

## **5. String**

Bài này chúng ta sẽ học cách sử dụng chuỗi trong JS

### **5.1. Khái niệm về chuỗi**

Ta có 2 cách tạo chuỗi trong JS như sau:

Cách 1:



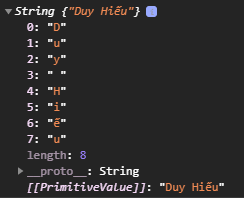
Kết quả là: 

Cách 2:



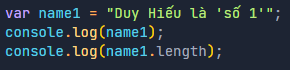
Kết quả là: 

Chúng ta nên sử dụng cách thứ 1. Vì khi sử dụng cách 2, biến name2 được khởi tạo bằng từ khoá (new) thì chuỗi tạo ra sẽ được định nghĩa là 1 object



*Kết quả của biến name2*

Cách xem độ dài chuỗi

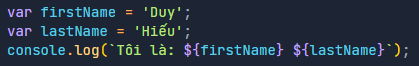


Kết quả là: 

Chúng ta sử dụng length để đếm số kí tự của chuỗi

### Template string trong ES6

Bắt đầu bằng `` và thêm các biến vào trong kí tự đó như sau `${variable}`



Kết quả là: 

### **5.2. Làm việc với chuỗi**

Sau đây sẽ là 1 số method của chuỗi mà chúng ta thường sử dụng:

* Find index
* Cut string
* Replace
* Convert to uppercase
* Convert to lowercase
* Trim
* Split

Đi cụ thể từng method



*Ví dụ*

#### **5.2.1. Find index**

Chúng ta có khá nhiều method có thể find index





Kết quả là: 

Method  cho phép chúng ta tìm kiếm giá trị ‘JS’ được khai báo, và kết quả sẽ trả lại vị trí chuỗi ‘JS’ đầu tiên trong chuỗi myString.





Kết quả là: 

Method  cho phép chúng ta tìm kiếm giá trị ‘JS’ được khai báo, tham số thứ 2 sẽ là vị trí chúng ta định nghĩa cho method bắt đầu tìm kiếm, và kết quả sẽ trả lại vị trí chuỗi ‘JS’ đầu tiên trong chuỗi myString từ vị trí chúng ta định nghĩa trước.





Kết quả là: 

Method  cho phép chúng ta tìm kiếm chuỗi ‘JS’ được khai báo, và kết quả sẽ trả lại vị trí chuỗi ‘JS’ cuối cùng trong chuỗi myString.





Kết quả là: 

Method  cho phép chúng ta kiểm tra xem chuỗi ‘JS’ được khai báo, nếu có sẽ trả về true và ngược lại.

#### **5.2.2. Cut string**





Kết quả là: 

Method  cho phép chúng ta cắt chuỗi, tham số thứ 1 là vị trí bắt đầu, tham số thứ 2 là vị trí kết thúc, và kết quả sẽ trả lại chuỗi mới đã được cắt.

#### **5.2.3. Replace**





Kết quả là: 

Method  cho phép chúng ta thay thế chuỗi mới từ chuỗi cũ, tham số thứ 1 là string cũ muốn thay thế, tham số thứ 2 là string mới, và kết quả sẽ trả lại chuỗi myString đã được thay thế. Tuy nhiên nếu không sử dụng biểu thức chính quy thì chúng ta chỉ  được 1 chuỗi ‘JS’ đầu tiên.



Sử dụng biểu thức chính quy ở chuỗi cũ như hình thì kết quả sẽ là: 

#### **5.2.4. Convert to uppercase**





Kết quả là: 

Method  cho phép chúng ta chuyển tất cả string sang chữ hoa.

#### **5.2.5. Convert to lowercase**





Kết quả là: 

Method  cho phép chúng ta chuyển tất cả chuỗi sang chữ thường.

#### **5.2.6. Trim**





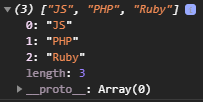
Kết quả là: 

Method  sẽ xoá những kí tự space dư thừa trong chuỗi

#### **5.2.7. Split**





Kết quả là: 

Method  cho phép chúng ta cắt chuỗi thành array bằng điểm chung, và kết quả trả lại cho chúng ta 1 array bao gồm các phần tử đã cắt.

## **6. Number**

### **6.1. Khái niệm về số**

Trong JS có 2 loại số:

* Các số thông thường trong JS được lưu trữ dưới dạng 64 bit IEEE-754 (còn được gọi là các số phẩy động). Trong hầu hết các trường hợp khi làm việc với Javascript là chúng ta sử dụng loại số này.
* Số BigInt là loại số sử dụng để biểu thị số nguyên có độ dài tùy ý.

Ta cũng có 2 cách khởi tạo số:

Cách 1:



Kết quả là: 

Cách 2:



Kết quả là: 

Tương tự như string, ta nên khởi tạo bằng cách thứ 1 bởi vì ở cách thứ 2 thì biến num được khởi tạo sẽ được định nghĩa là object

### **6.2. Làm việc với số**

Đây là 1 số method của số mà chúng ta thường sử dụng:

* Check integer
* To string
* To fixed

Đi cụ thể từng method:

#### **6.2.1. Check integer**





Kết quả là: 

Method  cho phép chúng ta kiểm tra xem 1 biến có phải là kiểu số nguyên hay không. Nếu đúng sẽ trả về true và ngược lại.

#### **6.2.2. To string**





Kết quả là: 

Method  cho phép chúng ta biến number thành string. Đây là number  và còn đây là string . Console cho kết quả khác màu để chúng ta nhận biết.

#### **6.2.3. To fixed**





Kết quả là: 

Method  cho phép chúng ta làm tròn đến chữ số mà chúng ta yêu cầu. tham số truyền vào sẽ là vị trí số thập phân mà chúng ta muốn làm tròn.

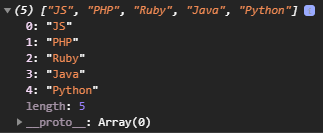
## **7. Array**

### **7.1. Khái niệm về mảng**

Có 2 cách để khởi tạo mảng:

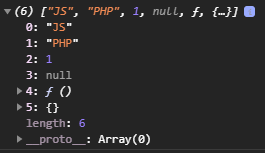
Cách 1:



Kết quả là: 

Cách 2:



Kết quả là: 

Chúng ta có thể truy xuất từng phần tử trong array bằng cách gọi đến index của phần tử đó trong mảng.



Kết quả là: 

### **7.2. Làm việc với mảng**

Đây là 1 số method của mảng mà chúng ta thường sử dụng

* check array
* toString
* pop
* push
* shift
* unshift
* splice
* concat
* slice

Đi cụ thể từng method



*Ví dụ*

#### **7.2.1. Check array**





Kết quả là: 

Method  cho phép chúng ta kiểm tra xem biến lang có phải là mảng không. Nếu đúng sẽ trả lại true và ngược lại.

#### **7.2.2. ToString**





Kết quả là: 

Method  cho phép chúng ta biến array thành string.

#### **7.2.3. Pop**





Method  cho phép chúng ta xoá phần tử cuối mảng và trả về phần tử đã xoá.

#### **7.2.4. Push**





Method  cho phép chúng ta thêm phần tử vào cuối mảng và trả về độ dài mới của mảng. Tham số truyền vào trong method là các phần tử mà chúng ta muốn thêm mới vào mảng.

#### **7.2.5. Shift**





Method  cho phép chúng ta xoá phần tử đầu mảng và trả về phần tử đã xoá.

#### **7.2.6. Unshift**





Method  cho phép chúng ta thêm phần tử vào đầu mảng và trả về độ dài mới của mảng. Tham số truyền vào trong method là các phần tử mà chúng ta muốn thêm mới vào mảng.

#### **7.2.7. Splice**





Method  cho phép chúng ta thay thế 1 hoặc 1 số phần tử của mảng bằng 1 hoặc 1 số phần tử khác. Method này sẽ gồm 3 tham số cơ bản, tham số thứ nhất là vị trí bắt đầu thay thế. Tham số thứ 2 là số phần tử sẽ bị loại bỏ, tính từ vị trí bắt đầu thay thế. Tham số thứ 3 sẽ là danh sách các phần tử mà chúng ta muốn thay thế bắt đầu từ vị trí bắt đầu thay thế.

#### **7.2.8. Concat**





Method  cho phép chúng ta nối array với array.

#### **7.2.9. Slice**





Method  cho phép trích xuất 1 phần tử của mảng. Method này gồm 2 tham số. Tham số thứ nhất là vị trí bắt đầu trích xuất. Tham số thứ 2 là vị trí kết thúc trích xuất. Giả sử (1,3) thì các phần tử được trích xuất sẽ là 1 và 2 (không bao gồm 3).

### **7.3. Array method**



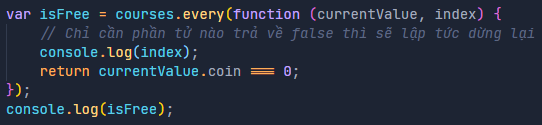
*Ta có 1 array chứa nhiều object sau*

#### **7.3.1. forEach**



Giống với vòng lặp thì method  duyệt từng phần tử của mảng. Đối số truyền vào của method  là 1 function. Function này được gọi là callback. Tham số trong function bao gồm currentValue (value của phần tử đang được duyệt) và index (index của phần tử đang được duyệt).

#### **7.3.2. every**

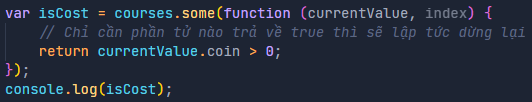


Method  cho phép chúng ta kiểm tra xem các phần tử thoả mãn 1 điều kiện mà chúng ta đặt ra. Tham số trong function bao gồm currentValue (value của phần tử đang được duyệt) và index (index của phần tử đang được duyệt).

* Trả về true nếu tất cả phần tử thoả mãn điều kiện.
* Trả về false nếu 1 trong số tất cả phần tử không thoả mãn điều kiện.

Lưu ý: cần phải return điều kiện mà chúng ta đặt ra.

#### **7.3.3. some**

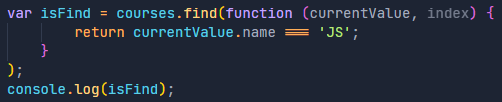


Method  thì ngược lại với method . Tham số trong function bao gồm currentValue (value của phần tử đang được duyệt) và index (index của phần tử đang được duyệt).

* Trả về true nếu 1 trong tất cả phần tử thoả mãn điều kiện.
* Trả về false nếu tất cả phần tử thoả mãn điều kiện.

Lưu ý: cần phải return điều kiện mà chúng ta đặt ra.

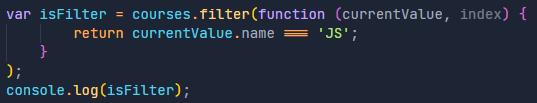
#### **7.3.4. find**



Đúng như cái tên, method  tìm kiếm phần tử trong mảng và trả lại 1 object. Tuy nhiên chỉ trả về phần tử đầu tiên mà nó tìm được. Nếu tìm được phần tử thoả mãn thì nó sẽ trả lại phần tử đó, nếu không nó sẽ trả về undefined. Tham số trong function bao gồm currentValue (value của phần tử đang được duyệt) và index (index của phần tử đang được duyệt).

Lưu ý: cần phải return điều kiện mà chúng ta đặt ra.

#### **7.3.5. filter**



Giống với method , tuy nhiên method  sẽ trả lại 1 mảng các object tìm được ứng với điều kiện. Tham số trong function bao gồm currentValue (value của phần tử đang được duyệt) và index (index của phần tử đang được duyệt).

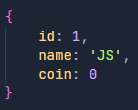
Lưu ý: cần phải return điều kiện mà chúng ta đặt ra.

#### **7.3.6. map**

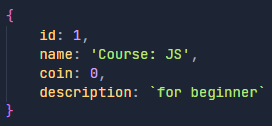
Trước tiên, chúng ta phải hiểu cách hoạt động của method .

Khi sử dụng method  sẽ trả lại cho chúng ta 1 array mới và số lượng phần tử bằng với số lượng phần tử có trong array. Giống với bao array method khác, method  cần có đối số truyền vào là function. Nó sẽ duyệt qua từng phần tử của array và trong method  chúng ta return về cái gì thì nó sẽ trả về số cái đó tương ứng với số lượng phần tử của array. Nếu không có return thì array trả về sẽ là số undefined tương ứng với số lượng phần tử của array.

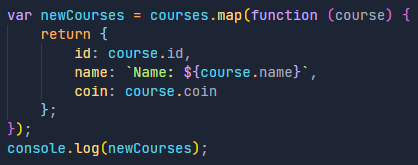
Ở method  chúng ta sẽ có 1 ví dụ để dễ dàng hiểu hơn. Theo như ví dụ ban đầu thì 1 object của chúng ta bao gồm các key và value như sau:



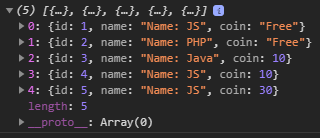
Bây giờ chúng ta sẽ sử dụng method  để biến object thành như thế này:



Sử dụng method  như sau:



Ở đây mỗi lần return nó sẽ thay đổi cho chúng ta phần tử mới như sau:



#### **7.3.7. reduce**

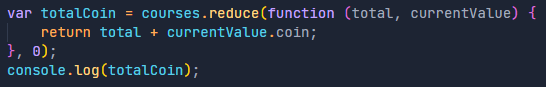
Chúng ta sử dụng method  khi muốn nhận về 1 giá trị duy nhất sau khi tính toán và xử lý các phần tử trong array. Method  có 2 đối số truyền vào:



Đối số thứ nhất là function, và đối số thứ hai là initialValue (ở đối số này chúng ta có thể sử dụng là number hoặc array) ở đây chúng ta sẽ tính tổng coin nên sẽ sử dụng là number.

Tiếp tục, trong function chúng ta lại phải truyền vào 2 đối số quan trọng đó là accumulator (biến tích trữ sẽ nhận từ giá trị initialValue) và currentValue (phần tử đang được duyệt trong array).

Thực hiện ví dụ trên, tính tổng coin các khoá học bằng method  như sau:

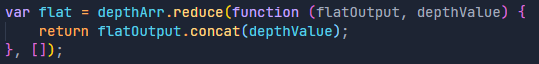


Lưu ý: return lại điều kiện mà chúng ta đặt ra.

Thêm 1 ví dụ về method  về flat array (làm phẳng mảng), có nghĩa là chúng ta sẽ biến array lồng array thành 1 array duy nhất.

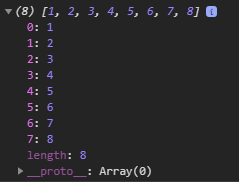


Ví dụ array lồng array



Ở đây, giá trị initialValue chúng ta sẽ truyền vào là [] để kết quả nhận lại được sẽ là 1 array. Điều kiện return chúng ta sử dụng method  để nối array với array sau mỗi lần duyệt.

Kết quả chúng ta thu lại được như sau:

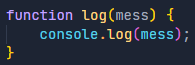


## **8. Function**

### **8.1. Tính chất về hàm**

* Không thực thi khi được định nghĩa
* Sẽ thực thi khi được gọi
* Có thể nhận tham số
* Có thể trả về 1 giá trị

### **8.2. Tham số và đối số hàm**



ở đây biến mess truyền vào function được gọi là tham số.



còn ‘test mess’ thì được gọi là là đối số.

Tuy nhiên người ta thường gộp chung lại và gọi là tham số.

* Chúng ta có thể truyền vào đối số bất kì các kiểu dữ liệu nào.
* Về tính private thì biến chúng ta sử dụng làm tham số chỉ được sử dụng ở trong hàm mà tham số đó được khai báo.

### **8.3. Arguments**

* Chỉ xuất hiện ở trong function.
* Có tính chất giống mảng.

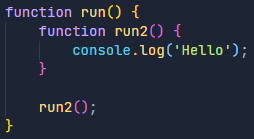


**8.4. Một số điều cần biết về hàm**

### Khi function được đặt trùng tên ???

Khi 2 function được đặt trùng tên thì function nào được định nghĩa sau sẽ ghi đè vào function được định nghĩa trước và nó sẽ thực thi.

### Định nghĩa hàm trong hàm ???

****

Và khi gọi function

****

Về tính private của hàm trong hàm thì vẫn giống với tham số trong hàm.

### **8.5. Các loại hàm**

Có 3 loại hàm:

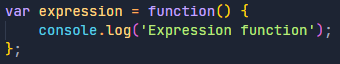
* Declaration function
* Expression function
* Arrow function (Sẽ học trong phần ES6)

Về Declaration function:

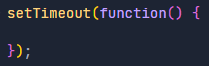
****

Chúng ta sẽ định nghĩa bằng từ khoá function đầu tiên và sau đó bắt buộc đặt tên cho function.

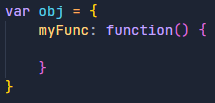
Về Expression function: chúng ta sẽ gặp loại function này ở nhiều dạng như sau



Function được khai báo từ 1 biến.



Function được khai báo trong 1 function built-in. function này đang ở dạng callback.



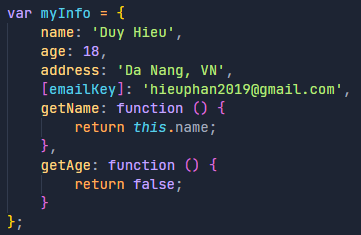
Function là 1 property trong object.

Tuy nhiên đối với Expression function thì chúng ta có thể đặt hoặc không đặt tên cho function, và mục đích đặt tên chỉ để chúng ta dễ hiểu.

## **9. Object**

### **9.1. Khái niệm về Object**

Object bao gồm các cặp key và value.



Trong trường hợp ta muốn thêm 1 cặp key và và value mới vào object đã tạo thì:

****

Tuy nhiên chúng ta cũng có thể tạo 1 key từ bên ngoài

****

Và thêm nó vào object như sau

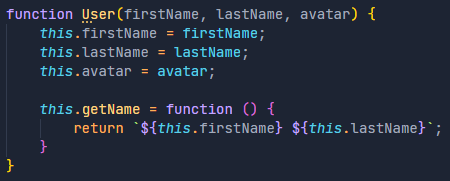
****

Để xoá 1 cặp key và value ta làm như sau



### **9.2. Object Constructor**

Object constructor là 1 bảng thiết kế của đối tượng.

****

Chúng ta sẽ tạo ra 1 object từ Object constructor trên như sau

****

Khởi tạo bằng từ khoá new đến Object constructor và lần lượt truyền các tham số.

Ở mỗi object được tạo ra từ Object constructor, chúng ta có thể tạo thêm các thuộc tính riêng cho nó

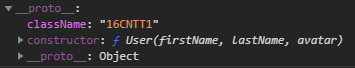


### **9.3. Object Prototype**

Constructor là 1 bảng thiết kế còn prototype là nguyên mẫu (nguyên liệu) để cấu thành nên 1 bảng thiết kế



Chúng ta sẽ tạo ra 1 prototype có key và value như trên. Những object được khởi tạo từ Object constructor này đều sẽ có prototype mà chúng ta vừa thêm vào. Tuy nhiên ở console chúng ta chỉ nhìn thấy prototype được thêm vào ở key \_\_proto\_\_ :



## **10. Date**

### **10.1. Khái niệm về Date**

Thường thì chúng ta sẽ sử dụng đối tượng Date để lấy ra những giá trị là thời gian.

****

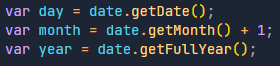
Khởi tạo 1 đối tượng date, kiểu dữ liệu của biến date lúc này là object.

Tuy nhiên nếu tạo như sau:



Biến date lúc này sẽ là string.

### **10.2. Sử dụng Date object**



ở biến month chúng ta phải +1 bởi vì method getMonth() sẽ trả lại cho chúng ta giá trị từ 0 -> 11.

****

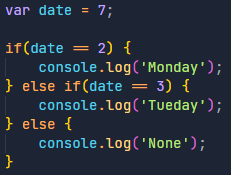
Kết quả lúc này là: 

Còn khá nhiều method khác mà chúng ta có thể tự tìm hiểu.

## **11. Branch Command**

### **11.1. If-else**

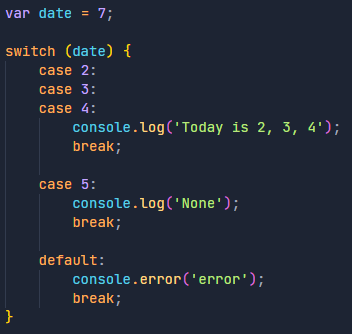
If-else là câu lệnh rẻ nhánh (câu lệnh điều kiện)



Chúng ta sử dụng if cho trường hợp điều kiện đầu tiên, else if cho các trường hợp điều kiện tiếp theo, và khi giá trị khai báo không rơi vào điều kiện nào thì nó sẽ thực thi ở else

### **11.2. Switch-case**

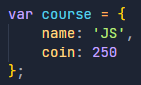
Switch-case là câu lệnh rẻ nhánh (câu lệnh điều kiện)



Khác với if-else chúng ta sẽ gán biến đã khai báo vào switch sau đó ở các case chúng ta sẽ định nghĩa các giá trị điều kiện. Trong mỗi case chúng ta phải break sau khi kết thúc điều kiện. Trong trường hợp giá trị khai báo không rơi vào điều kiện nào của case thì nó sẽ thực thi ở default.

### Lưu ý: switch-case sử dụng toán tử (===) để so sánh tuyệt đối các giá trị truyền vào, còn if-else chỉ sử dụng toán tử (==).

11.3. Toán tử 3 ngôi (Ternary Operator)



Sử dụng toán tử 3 ngôi như sau

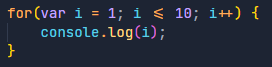


Điều kiện nếu đúng sẽ rơi vào sau dấu ? và nếu sai sẽ rơi vào sau dấu :

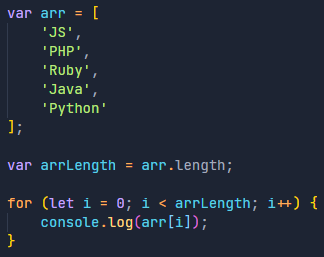
## **12. Loop**

### **12.1. Vòng lặp for**

Lặp với điều kiện đúng.



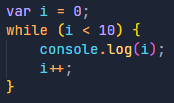
Trong vòng lặp for bao gồm 3 khối mã. Khối mã thứ nhất là khởi tạo biến đếm, khối mã thứ 2 là điều kiện dừng và khối mã cuối cùng là tăng hoặc giảm biến đếm. Áp dụng với duyệt mảng như sau:



Ta gán biến arrLength để tối ưu cho vòng lặp.

### **12.2. Vòng lặp while**

Lặp khi điều kiện đúng.

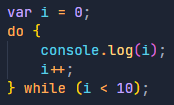


Trong vòng lặp while, trước tiên ta tạo biến đếm, tiếp theo sau while là điều kiện dừng và trong while là tăng hoặc giảm biến đếm. Áp dụng với duyệt mảng như sau:

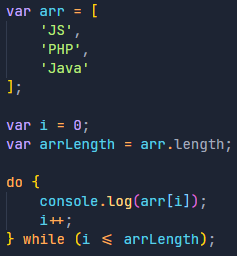


### **12.3. Vòng lặp do-while**

Lặp ít nhất 1 lần, sau đó lặp khi điều kiện đúng.



Về cơ bản khá giống với while, chúng chỉ khác nhau duy nhất. Đối với do-while dù điều kiện lặp như thế nào thì đoạn code vẫn được chạy ít nhất 1 lần còn nếu điều kiện thoả mãn thì sẽ tương tự như while. Áp dụng duyệt mảng như sau:

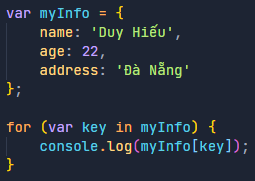


### **12.4. Vòng lặp for-in**

Lặp qua key của đối tượng.

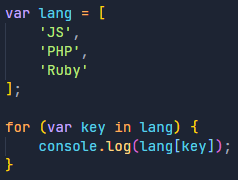
Vòng lặp này có chút khác biệt với các hàm phía trên. For-in có thể lặp được object, array hoặc string nhưng mục đích chủ yếu của vòng lặp này dùng dể lặp trong 1 object chứ không phải array. Số lượng vòng lặp sẽ tương ứng với số lượng thuộc tính của object.

#### **12.4.1. Đối với object**



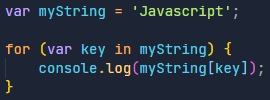
Ở vòng lặp for-in ta có 2 tham số truyền vào là key và object. Tham số key trong vòng lặp là các key trong object mà chúng ta khai báo, tham số object chính là tên object chúng ta khai báo. Lưu ý vòng lặp for-in sẽ trả về các key dạng string và để lấy các giá trị value tương ứng với key thì chúng ta gọi như hình.

#### **12.4.2. Đối với array**



Khác với object thì các key của array sẽ trả về dưới dạng thứ tự index của các phần tử trong array. Để lấy các giá trị value thì làm tương tự như hình. Lưu ý for-in không được khuyến khích sử dịng với những array mà thứ tự index của nó quan trọng.

#### **12.4.3. Đối với string**



Với kiểu string thì vẫn tương tự như array.

### **12.5. Vòng lặp for-of**

Lặp qua value của đối tượng.

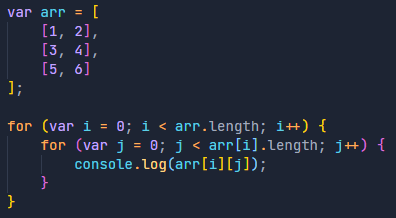
Vòng lặp for-of được ra mắt trong phiên bản ES6. Hàm này có thể sử dụng để duyệt phần lớn các đối tượng từ array, string, map, weakmap, set, ...



Trong vòng lặp for-of này có 2 tham số truyền vào. Tham số iterator là các value trong array mà chúng ta khai báo, tham số lang chính là array chúng ta khai báo.

### **12.6. Vòng lặp lồng vòng lặp (nested loop)**

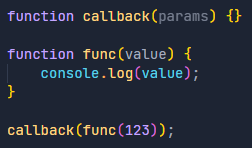
Trong trường hợp trong 1 mảng có nhiều mảng con thì chúng ta sẽ dùng vòng lặp lồng vòng lặp để lấy ra được các value.



## **13. Callback**

### **13.1. Khái niệm callback**

* Là hàm.
* Truyền qua đối số.
* Được gọi lại trong hàm nhận đối số.

****

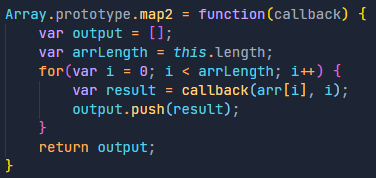
Chúng ta sẽ gọi function func() thông qua function callback() cho nên func() được gọi là callback function.

### **13.2. Hiểu thêm cách hoạt động của callback**

Chúng ta sẽ tiến hành viết lại array method  như sau:



Chúng ta khởi tạo 1 array method mới là  bằng array prototype

****

## **14. JSON**

### **14.1. Khái niệm về JSON**

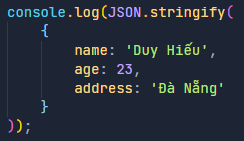
* Là 1 định dạng dữ liệu (string).
* Tên tiếng anh là JavaScript Object Notation (JSON).
* JSON thể hiện cho chúng ta được những kiểu dữ liệu: number, string, boolean, null, array, object.

### **14.2. Sử dụng đối tượng JSON**

Đối tượng JSON trong Javascript cung cấp cho chúng ta 2 method để convert như sau:

* Stringify: Từ Javascript types sang JSON.
* Parse: Từ JSON sang Javascript types.

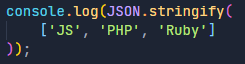
#### **14.2.1. Stringify**

****

Chúng ta stringify 1 object sang JSON như trên. Kết quả nhận được như sau:

****

Ở mỗi cặp key và value trong object sẽ được bỏ trong cặp “” để thể hiện cho chúng ta biết rằng đây là định dạng dữ liệu JSON.



Thử với array thì nhận được kết quả như sau:

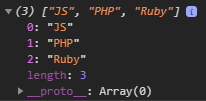


Mỗi thuộc tính trong array vẫn đều sẽ được bỏ trong cặp “”. Và chúng ta sử dụng method stringify của đối tượng JSON để convert từ JS types sang JSON.

#### **14.2.2. Parse**



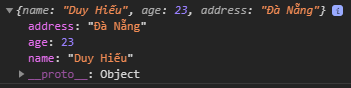
Kết quả nhận được như sau:



Vì JSON là dữ liệu string cho nên chúng ta sẽ bỏ trong cặp ‘’ và tất cả những phần tử trong array thì sẽ được bỏ trong cặp “”.



Thử với object thì nhận được kết quả như sau:



## **15. Promise**

### **15.1. Khái niệm về Promise**

* Đồng bộ (sync)
* Bất đồng bộ (async)

Promise sẽ có 3 trạng thái:

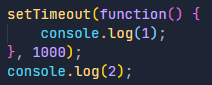
* Pending (đang xử lý)
* Fulfilled (đã hoàn thành)
* Rejected (đã bị từ chối)

Vậy đồng bộ là như thế nào ???

 Kết quả trả lại như sau: 

Cái nào gọi trước ra trước ra trước, gọi sau ra sau. Thế này gọi là sync.

Thế còn bất đồng bộ là như thế nào ???

 Kết quả trả lại như sau: 

Cái số 2 sẽ được gọi ra trước, sau 1000ms thì cái số 1 mới được gọi. Gọi là async. Và async thường sảy ra trong 1 số thao tác sau:

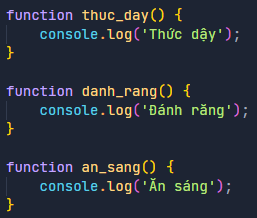
* 
* 
* 
* 
* 
* 

Và Javascript sử dụng khái niệm callback để xử lý những thao tác bất đồng bộ này.

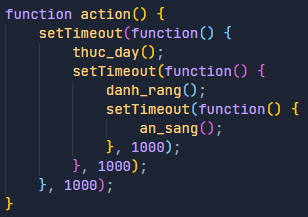
### **15.2. Pain**

Với sự phát triển như hiện nay của Javascript thì có một vấn đề cực kỳ nhức nhối đã được thể hiện với callback đó là callback hell. Callback hell hay còn được gọi là pyramid of doom, hadouken là cách code không tối ưu.

Lấy ví dụ, chúng ta có việc cần làm theo thứ tự A, B và C như sau:



Và callback hell sẽ như thế này

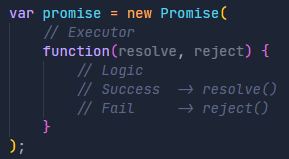
****

Code sẽ không tốt ưu.

### **15.3. Concept**

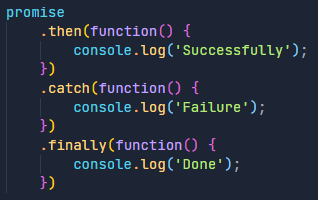
Để giải quyết vấn đề callback hell, có một lời hứa được hiện thực hoá, vâng tên của nó chính là Promise.

Promise giờ đã trở thành 1 trong những thư việc chuẩn của Javascript.

****

Khởi tạo đối tượng promise thông qua Object Contructor Promise, và đối tượng này nhận lại 1 function bao gồm 2 đối số là resolve và reject. Khi thao tác xử lý là thành công thì gọi đến resolve và ngược lại là reject.

Và sử dụng đối tượng promise như thế nào ?

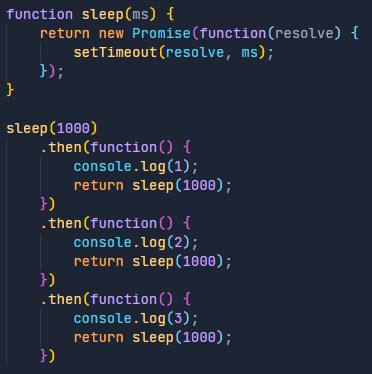
****

Đối tượng promise sẽ cung cấp cho chúng ta 3 method như trên. Method  sẽ thực thi khi đối tượng promise là resolve, method  sẽ thực thi khi đối tượng promise là reject, còn method  sẽ thực thi cho cả resolve và reject. Tuy nhiên method  rất ít khi được sử dụng.

### Promise là 1 khái niệm sinh ra để xử lý những thao tác bất đồng bộ, trước khi có promise thì người ta thường xử dụng callback, tuy nhiên callback thường say ra hiện tượng callback hell. Để tạo ra được 1 promise thì chúng ta sử dụng từ khoá new đến Promise constructor, ở trong nó chúng ta sẽ truyền vào 1 executor function gồm 2 tham số là resolve và reject ở dạng function. Resolve được gọi khi thao tác là thành công và reject thì ngược lại.

### **15.4. Chain**

Ở phần này chúng ta sẽ đi tìm hiểu tính chất chuỗi của Promise. Hiểu rõ hơn về tính chất này của Promise thì chúng ta làm 1 ví dụ như sau: xuất lần lượt các số từ 1 đến 3, mỗi số cách nhau 1000ms.



Đối với tham số resolve thì đối tượng của chúng ta có thể có nhiều method , nó thể hiện tính chất chuỗi của Promise. Khi method  đầu tiên return lại cái gì thì method  tiếp theo sẽ lấy cái đó làm data cho chính nó. Và kết quả như sau:

 Cứ sau mỗi 1000ms thì nó lại log ra một giá trị khác.

## **16. Fetch**

### **16.1. Khái niệm về fetch API**

**** cho phép tạo một network request và nó hoạt động theo Promise.

API(Appilication Programing Interface) là cổng giao tiếp giữa các phần mềm (URL).

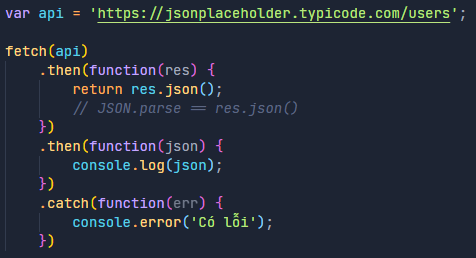
Cụ thể hơn: Back-end sẽ đưa cho chúng ta 1 cái API, sau đó chúng ta dùng **** để chuyển từ json sang object để sử dụng.

### **16.2. Sử dụng fetch**

Giả sử Back-end gởi cho chúng ta 1 cái API như sau:

****

Chúng ta sử dụng ****:

****

# **ES6**

## **1. Let & Const**

### **1.1 Tìm hiểu về khai báo biến**

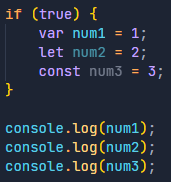
Chúng ta đã học ở phần JS về khai báo biến và hiểu về var, tuy nhiên ES6 đã bổ sung thêm 2 cách khai báo biến là let và const.

### **1.2. Sự khác nhau**

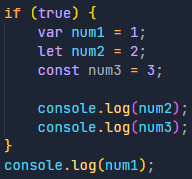
Sự khác nhau giữa var / let, const:

* scope
* hoisting

#### **1.2.1. Scope (phạm vi)**

 Kết quả: 

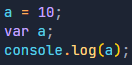
Tuy nhiên khi sửa lại như thế này:

 Kết quả: 

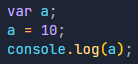
Có nghĩa là phạm vi truy cập biến của let và const là block scoped (trong phạm vi được khai báo) còn đối với var là globally scoped (toàn cục).

#### **1.2.2. Hoisting**

Trong ba loại khai báo biến thì chỉ có var được hổ trợ hoisting. Hoisting có nghĩa là đưa lên đầu.



Và khi trình đọc nó sẽ hoisting như sau:



Sự khác nhau giữa const / var, let:

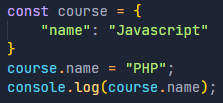
* Assignment

#### **1.2.3. Assignment (tái khai báo)**

Const khác với let và var là không thể tái khai báo (gán lại) được đối với biến có kiểu dữ liệu là: string, number, boolean, null và undefined.

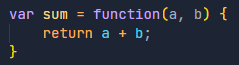
 Kết quả là: 

Đối với biến có kiểu là array, object hay function, tuy không thể tái khai báo nhưng chúng ta có thể cập nhật giá trị cho thuộc tính của biến.

 Kết quả là: 

## **2. Arrow Function**

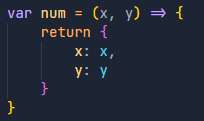
Arrow function thực chất cũng chỉ là một function bình thường tuy nhiên cách viết được rút gọn để linh hoạt hơn.



Rút gọn thành arrow funtion thành như sau:



Chúng ta bỏ chữ function ở đầu, chỉ giữa lại dấu ngoặc đơn và tham số truyền vào. Sau đó là 1 mũi tên (arrow). Nếu chúng ta không có {} thì mặc định ở sau có nghĩ là return. Tuy nhiên nếu chúng ta muốn return lại 1 object hay function thì sao.



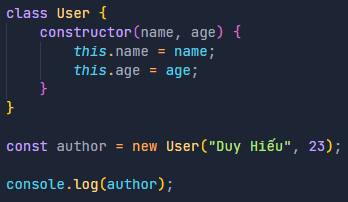
Chúng ta có thể viết như này.



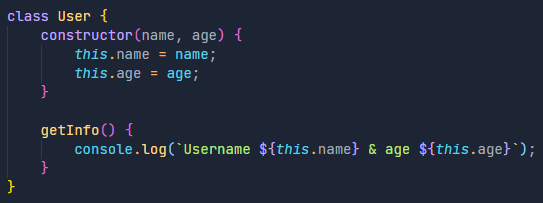
Hoặc bao hàm như này.

## **3. Classes**

Khái niệm và cách biểu diễn giữa classes và object constructor khá là giống nhau. Tuy nhiên hơi khác 1 chút và trông giống như hướng đối tượng (OOP).



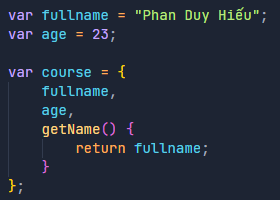
1 class chỉ có duy nhất 1 constructor, và constructor sẽ là phương thức chạy đầu tiên khi chúng ta khởi tạo 1 instance của class.



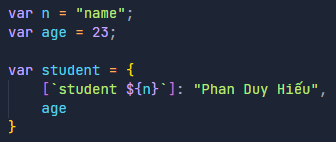
Trong class chúng ta còn có thể thêm các method class như trên hình.

## **4. Enhanced Object literals**

Bây giờ chúng ta đã có thể binding biến vào trong object và nó sẽ nhận luôn tên của biến đó là thuộc tính trong object. Chúng ta có thể định nghĩa key: value cho object và định nghĩa method cho object.



Hơn nữa chúng ta còn có thể định nghĩa key cho object dưới dạng biến.



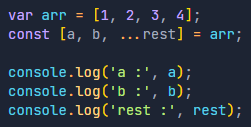
## **5. Destructuring**

### **5.1. Khái niệm về destructuring**

Destructuring là một cú pháp cho phép bạn gán các thuộc tính của một Object hoặc một Array. Điều này có thể làm giảm đáng kể các dòng mã cần thiết để thao tác dữ liệu trong các cấu trúc này.

### **5.2. Sử dụng destructuring**

#### **5.2.1. Destructuring Array**



Giải thích như sau: chúng ta khởi tạo 1 array có biến là a, b và …rest bằng với array của chúng ta khởi tạo. 2 biến a và b sẽ lần lượt lấy các giá trị tương ứng với index của nó và index trong array.

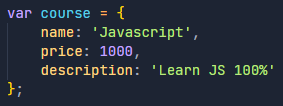
### Vậy còn …rest là gì?

…rest thì nó sẽ tương ứng với tất cả giá trị còn lại trong array của chúng ta khởi tạo. Trong ES6 người ta gọi đó là rest params hay rest ES6.

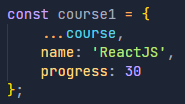
Kết quả trả lại như sau:



#### **5.2.2. Destructuring Object**



Chúng ta sẽ tạo ra 1 object mới kế thừa từ object cũ và thêm vào những property khác.



…course chính là đối tượng mà chúng ta kế thừa, những value nào chúng ta muốn thay đổi thì chỉ cần khai báo lại còn không muốn thay đổi thì không nhất thiết phải khai báo lại, ngoài ra chúng ta có thể thêm các cặp key và value mới mà không hề ảnh hưởng đến object mà chúng ta kế thừa trước đó,

Kết quả như sau:



## **6. Spread**

### **6.1. Khái niệm về spread**

Spread là một cách hữu dụng và ngắn gọn để dùng trong các thao tác với array như: sao chép array, kết hợp array (hoặc object), truyền tham số array vào function. Tuy nhiên cách thể hiện của spread khá giống rest params (…)

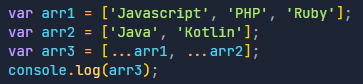
### **6.2. Sử dụng spread**

#### **6.2.1. Truyền tham số array vào function**



Khi chúng ta truyền tham số array vào function như trên thì toán tử spread sẽ bỏ cặp [], lúc này spread đã mở rộng array từ 3 phần tử thành 3 tham số riêng biệt.

#### **6.2.2 Kết hợp array**



Nếu chưa biết về spread thì trước đó chúng ta thường sử dụng array method . Và đây cũng có thể gọi là sao chép array hoặc thêm phần tử vào array. Kết hợp object thì chúng ta cũng làm tương tự.

# **DOM HTML**

## **1. DOM Object**

****

Gồm 3 thành phần chính:

1. Element
2. Attribute
3. Text

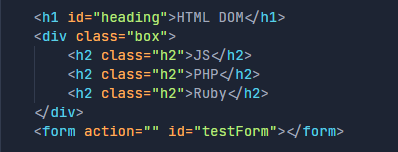
Đến với thành phần đầu tiên là:

### **1.1. Element**

Để get ra được những cái node trong DOM thông qua Element thì ta get:

* id
* class
* tag name
* CSS selector
* HTML collection

Ta có ví dụ sau:



*File html minh họa ví dụ*

Get qua id như sau:



‘heading’ ở đây là id của 1 element ở file html. Gọi ra console trả trực tiếp element

Get qua class như sau:



‘h2’ ở đây là class của 1 element ở file html. Tuy nhiên khi gọi ra console biến x2 thì sẽ trả lại cho chúng ta 1 phần tử HTMLCollection (tính chất vẫn là array) chứa các element có class tên là ‘h2’ mà chúng ta đã gọi đến.

Get qua tag như sau:



‘h2’ ở đây là element của html. Tương tự như class khi gọi ra console thì sẽ trả lại cho chúng ta 1 phần tử HTMLCollection chứa tất cả các element h2 được gọi.

Get qua CSS Selector như sau:



Bởi vì class ‘h2’ là con của class ‘box’ nên chúng ta sẽ truy cập như CSS Selector. Gọi ra console trả trực tiếp element. Giả sử trong class cha ‘box’ có nhiều class con ‘h2’ thì ta sẽ truy cập như thế nào?



Chúng ta có thể thay đổi số 1 thành số bất kì tùy thuộc vào thứ tự class mà chúng ta muốn chọn. Ví dụ ở trên là đang truy cập vào class con ‘h2’ thứ nhất.

Tuy nhiên nếu chúng ta muốn lấy tất cả class con ‘h2’ thì chúng ta sẽ sử dụng như sau:



Trong trường hợp này thì khi gọi ra console thì nó sẽ trả lại cho chúng ta 1 danh sách NodeList (tính chất vẫn là array) tương tự như HTMLCollection.

Get qua HTMLCollection như sau:

Đối với element form thì nó sẽ được quy là 1 HTMLCollection. Vì thế cách lấy form còn đơn giản hơn.



‘testForm’ ở đây chính là id của form ở file html.

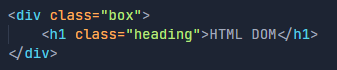
Lưu ý: có 1 tricks để sử dụng các method trên như sau:

* Hai method: getElementById và querySelector có dạng số ít nên chỉ trả ra console 1 kết quả là 1 element.
* Các method còn lại có dạng số nhiều nên console sẽ trả lại 1 mảng các kết quả.

### **1.2. Attribute**

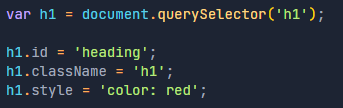
Set được attribute cho 1 element ta có 2 cách.

Ta có ví dụ sau:



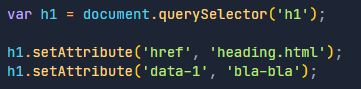
File html minh họa ví dụ

Cách 1:



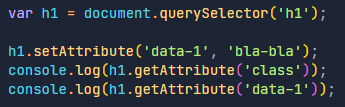
Với cách này chúng ta có thể truy cập trực tiếp vào những attribute có sẵn đã được định nghĩa trước như: id, class, … . Tuy nhiên có 2 số attribute không hỗ trợ cho element h1 như ‘href’ thì nó sẽ không cho phép set 1 attribute mới. Vậy làm sao ta có thể set cho element này.

Cách 2:



Với cách này chúng ta có thể xử lý được vấn đề ở cách 1. Hơn nữa chúng ta có thể thêm 1 số attribute tự tạo mà html không hổ trợ. Chúng ta phải chú ý cách dùng này, setAttribute có 2 đối số truyền vào, đối số thứ nhất là tên attribute mà chúng ta muốn add, đối số thứ 2 là giá trị của attribute.

Get attribute của element:



Chúng ta đều có thể get attribute qua attribute đã được định nghĩa trong element và cả attribute được tạo qua JS.

### **1.3. Text**

Chúng ta có 2 cách để lấy được text trong element như sau

Ta có ví dụ sau:



File html minh hoạ ví dụ

Cách 1: innerText



Và để thay đổi nội dung của text thì chúng ta làm như sau:



Tuy nhiên text chỉ thay đổi khi file JS được thực thi.

Cách 2: textContent



Khá giống với innerText, và để thay đổi nội dung của text thì chúng ta làm như sau:



Vẫn giống với innerText, text chỉ thay đổi khi file JS được thực thi.

### Vậy sự khác nhau giữa innerText và textContent là gì ???



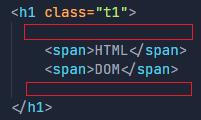
File html minh hoạ ví dụ

* Sử dụng innerText:

 Kết quả là nó sẽ bỏ qua các element chỉ lấy text

* Sử dụng textContent:

 Kết quả theo như trên file minh hoạ

 Kết quả là nó vẫn sẽ bỏ qua các element, tuy nhiên các phần đỏ (text node) vẫn được lấy

Đó chính là sự khác nhau giữa innerText và textContent.

* innerText sẽ trả ra các giá trị (bỏ qua các element) mà chúng ta nhìn thấy trên Browser.
* textContent sẽ trả ra tất cả giá trị (bỏ qua các element) kể cả khi chúng ta không nhìn thấy trên Brower nếu chúng được khai báo trong file HTML.

Một số text property:

Thêm element node: innerHTML



Khi gán giá trị có element thì giá trị đó sẽ gọi là element node, nếu không có element thì giá trị sẽ là text node. Hơn nữa nó còn có thể thêm attribute node